1. การเรียงข้อมูล (Sort)

บทเรียนนี้จะกล่าวถึงการเรียงข้อมูล (sorting) โดยเน้นเฉพาะในส่วนที่ใช้ในการเขียนโปรแกรมเพื่อการแข่งขัน อย่างไรก็ตามเนื้อหาเรื่องการเรียงลำดับของวิทยาศาสตร์คอมพิวเตอร์นั้นกว้างและครอบคลุมมากกว่าบทเรียนนี้

สำหรับการเขียนโปรแกรมเพื่อการแข่งขันแล้ว ข้อมูลที่นำมาเรียงลำดับนั้นจะถูกนำมาบรรจุในโครงสร้างข้อมูล จากนั้นทำการเรียงลำดับโดยใช้ฟังก์ชันที่เขียนเอง หรือฟังก์ชันมาตรฐานจาก STL การเขียนโปรแกรมเพื่อเรียงลำดับข้อมูลนั้นต้องคำนึง 3 ประการต่อนี้

1. ข้อมูลที่นำมาเรียงเป็นข้อมูลประเภทใด ชนิดใด ข้อมูลมีหลายประเภท ทั้งข้อมูลพื้นฐานอย่าง (เช่น int, float, double, char, bool), ข้อมูลที่กำหนดโดย STL class (เช่น pair, tuple, string, bitset), ข้อมูลแบบ user-defined (struct และ class) รวมถึงข้อมูลที่เป็นโครงสร้างข้อมูลแบบอาร์เรย์ หรือ STL container (เช่น vector, list, stack, queue, map, set)
2. ข้อมูลที่นำมาเรียงเก็บในโครงสร้างข้อมูลชนิดใด (อาร์เรย์, STL container, user-defined class)
3. การเรียงลำดับใช้สมบัติใดของข้อมูล ด้วยเกณฑ์อะไร

การเรียงลำดับข้อมูลที่เก็บในอาร์เรย์

การเรียงลำดับข้อมูลที่บรรจุในอาร์เรย์ 1 มิติสามารถทำได้โดยใช้ฟังก์ชันของ STL ชื่อ sort( ) ซึ่งเป็นฟังก์ชันที่รับค่าพารามิเตอร์ 2 หรือ 3 ตัว โดยที่พารามิเตอร์ 2 ตัวแรก (สมมุติว่าชื่อ first และ last) first ระบุตำแหน่งในหน่วยความจำของข้อมูลตัวแรก และ last ระบุตำแหน่งในหน่วยความจำหลังข้อมูลตัวสุดท้าย ส่วนพารามิเตอร์ตัวที่ 3 เป็นฟังก์ชันที่รับค่าพารามิเตอร์ 2 ตัว (สมมุติว่าชื่อ a และ b) ฟังก์ชันจะคืน bool ที่ระบุว่า a มีลำดับก่อนหน้า b หรือไม่

โปรแกรม 1.1 แสดงการใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด int (primitive type) และ string (ข้อมูลที่กำหนดโดย STL class) บรรทัดที่ 18 และ 20 แสดงการใช้ฟังก์ชัน sort( ) เรียงลำดับแบบ `ascending` (ตัวเลขเรียงจากน้อยไปมาก และตัวอักขระเรียงแบบ lexicographical) ซึ่งเป็นแบบปริยาย หากต้องการให้เรียงแบบ `descending` ต้องให้พารามิเตอร์ตัวที่ 3 คือฟังก์ชัน greater<T>( ) ดูตัวอย่างบรรทัดที่ 24 และ 26

ฟังก์ชัน sort( ) มี time complexity เป็น $O(N \log N)$ เมื่อ N เป็นจำนวนข้อมูลที่เรียงลำดับ

โปรแกรมที่ 1.1 การใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด int และ string

```
1.  #include<iostream>
2.  #include<algorithm>   // sort
3.  #include<string>      // string
4.  using namespace std;
5.  // Forwards
6.  template <typename T>
7.  void printArray(const T arr[], const int &n);
8.  int main(){
9.      int N = 5;
10.     int numbers[] = {6,2,4,6,-2};
11.     string pets[] = {"dog","bird","ferret","cat","fish"};
12.
13.     cout << "Before sort\n";
14.     printArray(numbers, N);
15.     printArray(pets, N);
16.
17.     cout << "\nAfter sort(ascending)\n";
18.     sort(numbers, numbers+N);
19.     printArray(numbers, N);
20.     sort(pets, pets+N);
21.     printArray(pets, N);
22.
23.     cout << "\nAfter sort (descending)\n";
24.     sort(numbers, numbers+N, greater<int>());
25.     printArray(numbers, N);
26.     sort(pets, pets+N, greater<string>());
27.     printArray(pets, N);
28.     cout << endl;
29.     return 0;
30. }
31. // print array of primitive and string
32. template <typename T>
33. void printArray(const T arr[], const int &n){
34.     for(int i=0; i<n; i++)
35.         cout << arr[i] << " ";
36.     cout << "\n";
37. }
```

ผลลัพธ์

```
Before sort
6 2 4 6 -2
dog bird ferret cat fish
After sort(ascending)
-2 2 4 6 6
bird cat dog ferret fish
After sort (descending)
6 6 4 2 -2
fish ferret dog cat bird
```

โปรแกรม 1.2 แสดงการใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด pair ข้อมูลชนิดนี้เก็บค่า 2 ค่า (เรียกว่า first และ second เมื่อเรานำฟังก์ชัน sort( ) มาใช้กับข้อมูลที่เก็บหลายค่าแบบนี้ เราต้องระบุว่าจะใช้ค่า first หรือ second สำหรับเรียงข้อมูล โดยปริยายจะใช้ค่า first (ดูบรรทัดที่ 25 และ 29) หากต้องการให้ใช้ค่า second ในการเรียงลำดับ (ดูบรรทัดที่ 33 และ 37) เราต้องเขียนฟังก์ชันที่ใช้เป็นพารามิเตอร์ตัวที่ 3 ของฟังก์ชัน sort( ) ดังนี้

- ฟังก์ชัน comparePair2ndAscending( ) ให้ใช้ค่า second ในการเรียงลำดับแบบแบบ ascending
- ฟังก์ชัน comparePair2ndAscending( ) ให้ใช้ค่า second ในการเรียงลำดับแบบแบบ descending

โปรแกรมที่ 1.2 การใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด pair

```
1.   #include<iostream>
2.   #include<algorithm>   // sort
3.   #include<string>      // string
4.   using namespace std;
5.
6.   // Forwards
7.   template <typename T>
8.   void printPairArray(const T arr[], const int &n);
9.   bool comparePair2ndAscending(const pair<string,int> &a,
10.                               const pair<string,int> &b);
11.  bool comparePair2ndDescending(const pair<string,int> &a,
12.                                const pair<string,int> &b);
13.
```

```
14.  int main(){
15.      int N = 5;
16.      pair<string,int> persons[] = {make_pair("Emily", 18),
17.                                    make_pair("Daisy", 20),
18.                                    make_pair("April", 18),
19.                                    make_pair("Lizzy", 16),
20.                                    make_pair("Jenna", 19)};
21.      cout << "Before sort\n";
22.      printPairArray(persons, N);
23.
24.      cout << "\nSort Pair by 1st item (ascending)\n";
25.      sort(persons, persons+N);
26.      printPairArray(persons, N);
27.
28.      cout << "\nSort Pair by 1st item (descending)\n";
29.      sort(persons, persons+N, greater< pair<string,int> >());
30.      printPairArray(persons, N);
31.
32.      cout << "\nSort Pair by 2nd item (ascending)\n";
33.      sort(persons, persons+N, comparePair2ndAscending);
34.      printPairArray(persons, N);
35.
36.      cout << "\nSort Pair by 2nd item (descending)\n";
37.      sort(persons, persons+N, comparePair2ndDescending);
38.      printPairArray(persons, N);
39.      cout << endl;
40.      return 0;
41.  }
42.  // print array of pair containing primitive and string
43.  template <typename T>
44.  void printPairArray(const T arr[], const int &n){
45.      for(int i=0; i<n; i++)
46.          cout << "(" << arr[i].first << "," <<
47.               arr[i].second <<") ";
48.      cout << "\n";
49.  }
50.
51.  // compare function to ascending sort by 2nd in pair
52.  bool comparePair2ndAscending(const pair<string,int> &a,
53.                               const pair<string,int> &b){
54.         return a.second < b.second;
55.  }
56.
57.  // compare function to descending sort by 2nd in pair
58.  bool comparePair2ndDescending(const pair<string,int> &a,
59.                                const pair<string,int> &b){
60.         return a.second > b.second;
61.  }
62.
63.
```

ผลลัพธ์

```
Before sort
(Emily,18) (Daisy,20) (April,18) (Lizzy,16) (Jenna,19)
Sort Pair by 1st item (ascending)
(April,18) (Daisy,20) (Emily,18) (Jenna,19) (Lizzy,16)
Sort Pair by 1st item (descending)
(Lizzy,16) (Jenna,19) (Emily,18) (Daisy,20) (April,18)
Sort Pair by 2nd item (ascending)
(Lizzy,16) (Emily,18) (April,18) (Jenna,19) (Daisy,20)
Sort Pair by 2nd item (descending)
(Daisy,20) (Jenna,19) (Emily,18) (April,18) (Lizzy,16)
```

โปรแกรม 1.3 แสดงการใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด Student ซึ่งเป็นข้อมูลแบบ user-defined class สำหรับข้อมูลแบบ user-defined (class และ struct) เราต้องเขียนฟังก์ชันเพื่อ overload operator< (ตัวดำเนินการเปรียบเทียบน้อยกว่า) ฟังก์ชันนี้ต้องชื่อ operator< แต่จะเขียนเป็นฟังก์ชันของ class Student (แบบที่แสดงบรรทัดที่ 23-25) หรือเป็นฟังก์ชัน global (แบบที่แสดงบรรทัดที่ 32-37) ก็ได้ ถ้าเขียนเป็นฟังก์ชันของ class Student จะรับพารามิเตอร์ 1 ตัว แต่ถ้าเขียนฟังก์ชัน global ก็ต้องรับพารามิเตอร์ 2 ตัว

โปรแกรมที่ 1.3 การใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด Student ที่เป็น user-defined class

```
1.  #include<iostream>
2.  #include<algorithm>   // sort
3.  #include<string>      // string
4.  using namespace std;
5.  // Forwards
6.  template <typename T>
7.  void printStudentArray(const T arr[], const int &n);
8.  class Student{
9.      public:
10.     int id;
11.     string name;
12.     int scores[3];
13.     // constructor
14.     Student(int id, string name, int s0, int s1, int s2){
15.         this->id = id; this->name = name;
16.         this->scores[0] = s0; this->scores[1] = s1;
17.         this->scores[2] = s2;
18.     }
```

```
19.      int getTotalScore(){
20.          return scores[0]+scores[1]+scores[2];
21.      }
22.      // overload operator< to sort() by name (ascending)
23.      bool operator<(Student b){
24.          return this->name < b.name;
25.      }
26.      void printStudent() const{
27.          cout<<"("<<id<<","<<name<<"=>["<<
28.                scores[0]<<","<<scores[1]<<","<<scores[2]<<"])";
29.      }
30.  };
31.
32.  /*
33.  // overload operator< to sort() by name (ascending)
34.  bool operator<(Student a, Student b){
35.      return a.name < b.name;
36.  }
37.  */
38.
39.  int main(){
40.      int N = 5;
41.      Student arr[] = { Student(1,"Emily", 10,9,10),
42.                        Student(2,"Daisy",10,10,10),
43.                        Student(3,"April",8,8,8),
44.                        Student(4,"Lizzy", 7,8,9),
45.                        Student(5,"Jenna", 9,8,7) };
46.      cout << "Before sort\n";
47.      printStudentArray(arr,N);
48.      sort(arr,arr+N);
49.      cout << "After sort by name\n";
50.      printStudentArray(arr,N);
51.      return 0;
52.  }
53.  // print array of class Student
54.  template <typename T>
55.  void printStudentArray(const T arr[], const int &n){
56.      for(int i=0; i<n; i++){
57.          arr[i].printStudent();
58.          cout << "\n";
59.      }
60.  }
```

ผลลัพธ์

```
Before sort
(1,Emily=>[10,9,10])
(2,Daisy=>[10,10,10])
(3,April=>[8,8,8])
(4,Lizzy=>[7,8,9])
(5,Jenna=>[9,8,7])
After sort by name
(3,April=>[8,8,8])
(2,Daisy=>[10,10,10])
(1,Emily=>[10,9,10])
(5,Jenna=>[9,8,7])
(4,Lizzy=>[7,8,9])
```

การเรียงลำดับข้อมูลที่เก็บใน STL container

ฟังก์ชัน sort( ) สามารถนำมาใช้กับ STL container ใดก็ได้ที่มี random access iterator ซึ่งก็คือ vector, deque และ array การเรียกใช้งานฟังก์ชัน sort( ) กับข้อมูลที่บรรจุใน vector, deque และ array เหมือนกับข้อมูลที่บรรจุในอาร์เรย์ 1 มิติ ยกเว้นแต่ตำแหน่งของข้อมูลตัวแรกและตัวสุดท้าย เราต้องระบุโดยใช้ iterator ของ container ผ่านฟังก์ชัน begin( ) และ end( ) ตามลำดับ

นอกจากนั้นแล้วสำหรับ STL container อย่าง forward_list (single linked list) และ list (double linked list) ซึ่งแม้จะไม่มี random access iterator ก็ยังมีฟังก์ชัน sort( ) เป็นของตนเองที่สามารถใช้เรียงข้อมูลได้ด้วย time complexity $O(N \log N)$ เช่นกัน

โปรแกรม 1.4 แสดงการใช้ฟังก์ชัน sort( ) กับข้อมูลแบบ primitive และ string ที่เก็บใน vector, deque และ array

โปรแกรมที่ 1.4 การใช้ฟังก์ชัน sort( ) กับข้อมูลที่เก็บใน vector, deque และ array

```
1.  #include<iostream>
2.  #include<vector>     // vector
3.  #include<deque>      // deque
4.  #include<array>      // array
5.  #include<algorithm>  // sort
6.  #include<string>     // string
7.  using namespace std;
8.
9.  // Forwards
10. template <typename T>
11. void printContainer(const T &con);
12.
13. int main(){
14.     vector<int> numbers = {6,2,4,6,-2};
15.     deque<string> pets = {"dog","bird","ferret","cat","fish"};
16.     array<double, 5> values = {2.5, 1.5, 3.5, 5.5, 4.5};
17.
18.     cout << "Before sort\n";
19.     printContainer(numbers);
20.     printContainer(pets);
21.     printContainer(values);
22.     cout << "\nAfter sort(ascending)\n";
23.     sort(numbers.begin(), numbers.end());
24.     printContainer(numbers);
25.     sort(pets.begin(), pets.end());
26.     printContainer(pets);
27.     sort(values.begin(), values.end());
28.     printContainer(values);
29.
30.     cout << "\nAfter sort (descending)\n";
31.     sort(numbers.begin(), numbers.end(), greater<int>());
32.     printContainer(numbers);
33.     sort(pets.begin(), pets.end(), greater<string>());
34.     printContainer(pets);
35.     sort(values.begin(), values.end(), greater<double>());
36.     printContainer(values);
37.     cout << endl;
38.     return 0;
39. }
40.
41. // print vector & deque of primitive and string
42. template <typename T>
43. void printContainer(const T &con){
44.     for(auto &i : con)
45.         cout<<i<< " ";
46.     cout << "\n";
47. }
```

ผลลัพธ์

```
Before sort
6 2 4 6 -2
dog bird ferret cat fish
2.5 1.5 3.5 5.5 4.5
After sort(ascending)
-2 2 4 6 6
bird cat dog ferret fish
1.5 2.5 3.5 4.5 5.5
After sort (descending)
6 6 4 2 -2
fish ferret dog cat bird
5.5 4.5 3.5 2.5 1.5
```

โปรแกรมที่ 1.5 การใช้ฟังก์ชัน sort( ) ในการเรียงข้อมูลชนิด Student ที่เป็น user-defined class

```
1.  #include<iostream>
2.  #include<vector>     // vector
3.  #include<algorithm>  // sort
4.  #include<string>     // string
5.  using namespace std;
6.  // Forwards
7.  template <typename T>
8.  void printStudentContainer(const T &con);
9.  class Student{
10.     public:
11.     int id;
12.     string name;
13.     int scores[3];
14.     // constructor
15.     Student(int id, string name, int s0, int s1, int s2){
16.         this->id = id; this->name = name;
17.         this->scores[0] = s0; this->scores[1] = s1;
18.         this->scores[2] = s2;
19.     }
```

```
20.       int getTotalScore(){
21.           return scores[0]+scores[1]+scores[2];
22.       }
23.       // overload operator< to sort() by ID (descending)
24.       bool operator<(Student b){
25.           return this->id > b.id;
26.       }
27.       void printStudent() const{
28.           cout<<"("<<id<<","<<name<<"=>["<<
29.                 scores[0]<<","<<scores[1]<<","<<scores[2]<<"])";
30.       }
31.   };
32.
33.   int main(){
34.       vector<Student> vec ={ Student(1,"Emily", 10,9,10),
35.                              Student(2,"Daisy",10,10,10),
36.                              Student(3,"April",8,8,8),
37.                              Student(4,"Lizzy", 7,8,9),
38.                              Student(5,"Jenna", 9,8,7) };
39.       cout << "Before sort\n";
40.       printStudentContainer(vec);
41.       sort(vec.begin(),vec.end());
42.       cout << "After sort by ID (descending)\n";
43.       printStudentContainer(vec);
44.       return 0;
45.   }
46.   // print vector/array/deque of class Student
47.   template <typename T>
48.   void printStudentContainer(const T &con){
49.       for(auto &i : con){
50.           i.printStudent();
51.           cout << "\n";
52.       }
53.       cout << "\n";
54.   }
```

ผลลัพธ์

```
Before sort
(1,Emily=>[10,9,10])
(2,Daisy=>[10,10,10])
(3,April=>[8,8,8])
(4,Lizzy=>[7,8,9])
(5,Jenna=>[9,8,7])
```

```
After sort by ID (descending)
(5,Jenna=>[9,8,7])
(4,Lizzy=>[7,8,9])
(3,April=>[8,8,8])
(2,Daisy=>[10,10,10])
(1,Emily=>[10,9,10])
```

โปรแกรมที่ 1.6 การใช้ฟังก์ชัน sort( ) ในการเรียง vector ที่เก็บใน vector

```
1.  #include<iostream>
2.  #include<vector>     // vector
3.  #include<algorithm>  // sort, max_element( )
4.  using namespace std;
5.  // Forwards
6.  bool compBy2ndColumn(const vector<int> &a, const vector<int> &b);
7.  bool compByMaxColumn(const vector<int> &a, const vector<int> &b);
8.  void print2DVector(const vector< vector <int> > &vec);
9.  int main(){
10.     vector<vector<int>> data = {{0,1,2,9},
11.                                 {8,7,6,0},
12.                                 {7,6,4,5}};
13.     cout << "Before sort\n";
14.     print2DVector(data);
15.     cout << "After sort with 2nd column (ascending)\n";
16.     sort(data.begin(), data.end(),compBy2ndColumn);
17.     print2DVector(data);
18.     cout << "After sort with maximum in row (ascending)\n";
19.     sort(data.begin(), data.end(),compByMaxColumn);
20.     print2DVector(data);
21.     return 0;
22. }
23.
24. // function to sort()vector by element at index 1 (ascending)
25. bool compBy2ndColumn(const vector<int> &a, const vector<int> &b){
26.     return a[1] < b[1];
27. }
28.
29. // function to sort()vector by maximum element (ascending)
30. bool compByMaxColumn(const vector<int> &a, const vector<int> &b){
31.     int max_a = *max_element(a.begin(),a.end());
32.     int max_b = *max_element(b.begin(),b.end());
33.     return max_a < max_b;
34. }
```

```
35.  void print2DVector(const vector< vector <int> > &vec){
36.      for(auto &r : vec){       // O(MN)
37.          for(auto &c : r)
38.              cout << c << " ";
39.          cout << "\n";
40.      }
41.      cout << "\n";
42.  }
```

ผลลัพธ์

```
Before sort
0 1 2 9
8 7 6 0
7 6 4 5
After sort with 2nd column (ascending)
0 1 2 9
7 6 4 5
8 7 6 0
After sort with maximum in row (ascending)
7 6 4 5
8 7 6 0
0 1 2 9
```

โปรแกรม 1.6 แสดงการใช้ฟังก์ชัน sort( ) ในการเรียงลำดับ vector ที่ถูกเก็บใน vector ซึ่งเป็นการเก็บข้อมูลที่เหมือนกับอาร์เรย์ 2 มิตินั้นเอง การเรียงลำดับในตัวอย่างนี้เหมือนกับการเรียงลำดับของแถวในอาร์เรย์ 2 มิติ ซึ่งเราสามารถเขียนฟังก์ชันที่ใช้ในการเปรียบเทียบที่นำสมบัติบางประการของแต่ละแถวมาเป็นตัวเทียบ เช่น ค่าที่คอลัมน์ที่ 2 หรือค่าสูงสุดในแถวนั้น

นอกจากฟังก์ชัน sort( ) แล้ว STL ยังมีฟังก์ชัน stable_sort( ) ที่สามารถเรียงข้อมูลโดยรักษาลำดับเดิมของข้อมูลมีซ้ำกัน ตัวอย่างเช่น ถ้าใช้ stable_sort( ) ในการเรียงข้อมูล 5 2 4 **1** 6 1 ซึ่งมีเลข 1 สองตัว (ตัวหนากับตัวขีดเส้นใต้ ผลลัพธ์ที่ได้จะเป็น **1** 1 2 4 5 6 (Time complexity $O(N\log^2 N)$)